IERG 4210 Tutorial 3 - Phase 2B

Part 1: Database, SQL, and Database Abstraction Layer

Xiao Yi modified according to previous TA **Siyue Xie**'s slides

yx019@ie.cuhk.edu.hk

Outline:

- Database
- Basic SQLs
- Database Abstraction Layer: PDO

1. Database

The database provides the functionality for storing and accessing data on the server. There are many types of databases, e.g., relational, NoSQL, NewSQL, etc. You are free to use any type of database for the assignments, but in this tutorial, we will only cover one relational database, i.e., SQLite.

Basically, the relational database contains table(s) for storing data, where each table consists of rows (records) and columns (attributes). For instance, we list two tables for products and categories.

- Products

PID	CATID	NAME	PRICE
1	1	iPhone 13	6799
2	1	iPhone 13 pro	8499
3	2	iPad Air	4799
4	2	iPad Pro	6399
5	3	MacBook Air	7799
6	3	MacBook Pro	9999

- Categories

CATID	NAME
1	Phone
2	Tablet
3	Laptop

Where `Products` lists all the items with the product's ID `PID`, the name `NAME`, the price `PRICE`, and the category it belongs - in the form of `CATID`. Moreover, `Categories` lists the name of each category and its `CATID`.

Specifically, we call `PID` in `Products` and `CATID` in `Categories` as **primary keys**, and `CATID` in `Products` as **foreign key**, which corresponds to `CATID` in `Categories`.

As previously mentioned, we will use **SQLite** in this tutorial for the following reasons:

- Public domain license
    - i.e., it is **free**
- **Lightweight** in design
    - Multiple processes can read at the same time; however, only one process can make changes at any moment in time
    - Best for apps (MobileApps/Simple WebApps)
- Supported by **multi-platforms**,
    - e.g., Windows, Linux, and Mac
    - Pre-installed in AWS VM
    - Built-in support since PHP 5
- Stores everything in a **single file**
    - Easy to embed, test, backup and transfer
- Simple access management
    - No user account management as in full-blown DBs like MySQL
    - Depending on the file access rights

Security Warning: Being the most important asset of any website, SQLite DB file **MUST NOT** be accessible by the public web (end-user); otherwise, it may raise serious security problems.

Suppose `/var/www/html` is where you host your public web pages; you should never put your DB file under this directory.

You can create a database named `cart.db` at `/var/www/` by the follwoing command:

```
sudo sqlite3 /var/www/cart.db
```

Then enable foreign-key support:

```
PRAGMA foreign_keys = ON;
```

And type `.quit` if you want to quit the SQLite command shell.

2. Basic SQLs

Since we already have two tables in mind, we want to create and manipulate these tables in the database.

2.1 Create the Table

1. Open the database just created

   ```
   sudo sqlite3 /var/www/cart.db
   ```

2. Create the `Categories` table

   ```
   CREATE TABLE CATEGORIES (
     CATID INTEGER PRIMARY KEY,
     NAME TEXT
   );
   ```

   Note: Primary key is unique and auto-increment by default.

3. Create the `Products` table

   ```
   CREATE TABLE PRODUCTS (
     PID INTEGER PRIMARY KEY,
     CATID INTEGER,
     NAME TEXT,
     PRICE REAL,
     FOREIGN KEY(CATID) REFERENCES CATEGORIES(CATID)
   );
   ```

4. Create an index for `CATID` to make related queries faster

   ```
   CREATE INDEX I1 ON PRODUCTS(CATID);
   ```

5. Check what you have created

   ```
   .schema
   ```

6. (If you did something wrong) delete the table

   ```
   DROP TABLE PRODUCTS;
   ```

2.2 Insert the Record

1. Insert a record into `Categories`

   ```
   INSERT INTO CATEGORIES VALUES (NULL, "Phone");
   ```

   Note: `NULL` for auto-increment primary key's value. In this case, since this is the first record in `Categories`, the `CATID` of "Phone" will be 1.

2. Insert a record into `Products`

    ```
    INSERT INTO PRODUCTS VALUES (NULL, 1, "iPhone 13", 6799);
    ```

    Or

    ```
    INSERT INTO PRODUCTS (CATID, NAME, PRICE)
      VALUES (1, "iPhone 13", 6799);
    ```

    Note: If the value of `CATID` does not exist in `Categories`, it will raise an error.

For more information about SQL `INSERT`: https://www.sqlite.org/lang_insert.html

2.3 Query the Record

Basically, we use the `SELECT` statements for querying records in tables, e.g., output all the records in the `Products` table:

```
SELECT * FROM PRODUCTS;
```

Moreover, you can add some constraints on the query by using `WHERE`, e.g., output all the products that belong to the "Phone" category:

```
SELECT * FROM PRODUCTS WHERE CATID = 1;
```

And if you want to find all phones with a price no more than 8000:

```
SELECT * FROM PRODUCTS WHERE CATID = 1 AND PRICE <= 8000;
```

For more information about SQL `SELECT`: https://www.sqlite.org/lang_select.html

2.4 Update the Record

If you want to update the record, you can use the `UPDATE` statement, e.g., update the price into 5999 of the product with `PID = 1`:

```
UPDATE PRODUCTS SET PRICE = 5999 WHERE PID = 1;
```

Or, if you want to decrease the price for all phone products by 1000:

```
UPDATE PRODUCTS SET PRICE = PRICE - 1000 WHERE CATID = 1;
```

For more information about SQL `UPDATE`: https://www.sqlite.org/lang_update.html

2.5 Delete the Record

You can use the `DELETE` statement to delete the record, e.g., if you want to delete the product with `PID = 1`:

```sql
DELETE FROM PRODUCTS WHERE PID = 1;
```

However, if you want to delete the "Phone" category in `Categories` like this:

```sql
DELETE FROM CATEGORIES WHERE NAME = "PHONE";
```

You may occur an error.

As the foreign key is on, a record with children cannot be deleted. In this case, the record `(1, "Phone")` in `Categories` still has a child `(1, 1, "iPhone 13", 6799)` in `Products`, thus it cannot be deleted. If you want to delete the record in `Categories`, delete all its children in `Products` first.

For more information about SQL `DELETE`: https://www.sqlite.org/lang_delete.html

3. Database Abstraction Layer: PHP Data Objects (PDO)

We can use the abstraction layer to connect and access different databases.

PHP provides the PDO for such purposes. It has the following advantages:

- **Coding Consistency**: Regardless of the DB, use the same set of code
- **Universal Interface**: Easy to switch database when the website grows
- **Performance**: PDO is written in C and compiled into PHP

3.1 PHO Installation

By default, it should be installed already; if not, you can use the following command to install:

```bash
sudo yum install php-pdo
```

If it does not work, you can try the followings:

- Check whether PDO is activated
    - `php.ini` →

      find ";extension=pdo_sqlite" →

      remove the ";" (uncomment) →

      restart Apache
- Optionally, backup your `/etc/php.ini` and `/etc/httpd/conf/httpd.conf`
- Reinstall all together PHP and Apache with `php-pdo`
- Restart Apache

3.2 PHP PDO

One of the important reasons to use PDO is that PDO can use prepared statements to prevent SQL injection. For instance, if we directly accept user input and use it to form the SQL statement, it may lead to data leaks or data loss.

If you want to know more about SQL injection, you can visit the following page:

SQL Injection: https://en.wikipedia.org/wiki/SQL_injection

We now provide some examples to interact with PDO:

1. Connect the database

    ```
    $db = new PDO('sqlite:../cart.db');
    ```

2. Enable foreign key

    ```
    $db->query('PRAGMA foreign_keys = ON;');
    ```

3. Prepare a statement

    ```
    $q = $db->prepare('SELECT * FROM PRODUCTS WHERE CATID = ?');
    ```

4. Bind the parameters

    ```
    $q->bindParam(1, $catid);
    ```

5. Execute the statement

    ```
    $q->execute();
    ```

For more information about PDO: https://www.php.net/manual/en/book.pdo.php

3.3 PHP Access Control

When a php file runs with an Apache server, the current user is Apache regardless of whom owns the file. Thus, you need to grant Apache permission to modify `cart.db` by running the following commands:

```
sudo chgrp apache /var/www
sudo chgrp apache /var/www/cart.db
sudo chmod 775 /var/www
sudo chmod 775 /var/www/cart.db
```